HZ BOOKS
华章IT

零基础学习TypeScript。
从基础知识到应用实战，包含大量实用案例。

實戰

胡桓铭 编著

Practical Guide to TypeScript

TypeScript 实战指南

机械工业出版社
China Machine Press

實戰

Practical Guide to TypeScript

TypeScript 实战指南

胡桓铭 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

TypeScript 实战指南 / 胡桓铭编著 . —北京：机械工业出版社，2019.4
（实战）

ISBN 978-7-111-62670-1

I. T… II. 胡… III. JAVA 语言 - 程序设计 - 指南 IV. TP312.8-62

中国版本图书馆 CIP 数据核字（2019）第 078601 号

TypeScript 实战指南

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：吴 怡
责任校对：殷 虹
印 刷：北京市荣盛彩色印刷有限公司
版 次：2019 年 5 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16
印 张：16.75
书 号：ISBN 978-7-111-62670-1
定 价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066
投稿热线：（010）88379604
购书热线：（010）68326294
读者信箱：hzit@hzbook.com

Preface 前　言

与 TypeScript 相遇，还是在 ThoughtWorks 工作的时候。那个时候，我们需要维护大量的前端遗留项目，需要与客户规划我们的人效，需要控制系统迭代带来的 bug 率。我们接手的项目往往缺乏严谨的注释和完整的代码说明文档，这导致在维护 JavaScript 遗留项目时，需要花费更多的时间去厘清参数及函数之间的关系，甚至需要用 debugger 逐层去观察值的变化。工作非常低效，但客户的需求又总是急迫的，这迫使我们去思考如何提升团队的工作效率。

这个时期也是 Facebook 开始推广 Flow 的时候。我们觉得添加静态类型应该是个非常不错的方向，也看了很多应用静态类型的成功案例。碰巧 Flow 对于遗留项目非常友好，你不需要为每个文件、每个函数、每行代码都添加类型，而只需要在你认为有必要的地方写上类型即可。所以我们很快进行了实验。

然而，我们在采用 Flow 后不久就发现了很多新产生的问题：

1）升级困难，配置复杂。尤其是在 React Native 项目中，经常会在升级后运行失败。

2）生态弱势。很多第三方库当时没有 Flow 的类型问题件。

3）难于上手。Flow 的气质更像考究的学院派风格，功能强大灵活，但对于新加入团队的人而言，其难度令人生畏。

于是，我们又将目光投向了 TypeScript。最初了解 TypeScript 是看到 Angular 团队在更新 Angular 2 时开始全面采用 TypeScript 代码。他们给出了这样两个理由：

1）TypeScript 明确了抽象。在大型工程项目中，我们希望模块之间的边界是使用接口定义的，而 JavaScript 不足以清晰表达类似的边界划分，Flow 也不能。而 TypeScript 可以定义接口，可以强制程序员去思考 API 的边界，去设计代码，而不只是编写代码，暴露代码的耦合。

2）TypeScript 可以使代码在一定程度上达到“Self-documenting”的效果。“Self-documenting”是一个非常有意思的概念，它强调的是代码本身具有自我说明的效果，而不是依

赖文档。TypeScript 有着非常严格的强类型表达，这迫使你在函数使用之前就必须标注好函数的入参和返回值类型。这样的强依赖使得函数本身表达清晰，同时也可以非常容易地推导出代码的依赖结构，进行重构。

之后，我们开始尝试在遗留项目中进行 TypeScript 重构，那是一种相见恨晚的感觉。从后期的数据来看，我们很有效地降低了 bug 率，同时支持项目的人效也得到了极大的提升。

这一段经历，使我重新开始思考关于语言静态类型的问题。在大型团队开发时，沟通的成本往往是极高的。这就是为什么在后端开发中，拥有静态类型的语言仍然占据主流，也是为什么 Python 在 3.5 版本中加入 Type Hint。显式的类型声明不仅有利于阅读，也有利于代码编辑器进行代码提示和依赖分析。

比如，在 Java 开发中，如果需要重构的话，依赖 IntelliJ IDEA 提供的函数重构功能，可以自动地对每一个依赖该函数的文件进行自动化重构。但这在 JavaScript 中是不可想象的，你只能使用全局文本搜索来修改函数名，这种操作非常原始，就像在现代战争中还拿着石锤向着敌方阵地冲刺一样。

这就是 TypeScript 为 JavaScript 生态带来的价值，也是为什么 Angular 和 Vue 都转向使用 TypeScript 进行重构。比起学术型的 Flow 而言，TypeScript 更像一门工程型的语言，它配置容易，上手快速，更适合在实战中使用，是一件非常称手的“兵器”。

我希望读者在使用 TypeScript 之前，能够对 TypeScript 有足够的了解。我结合 TypeScript 的官方手册与其他公开资料，整理了一些 TypeScript 基础的内容，就是本书的“基础篇”，最好粗略过一遍这部分内容。在“实战篇”中会提及这些内容，返回去再看时，反而能加深理解。

实际上，如何在实战中使用 TypeScript 反而是一个老大难的问题。这也是初学者更容易遇到的困难。“为什么手册读完了，官方实例也看了，我还是不会在 React 里写 TypeScript 呢？”这是我经常听到的反馈，希望本书能够很好地回答这类问题。

最后，非常感谢 2018 年年底的住院经历，因为无法完全被治愈，使得我开始重新思考生命与健康的问题，如果有机会我也非常想聊聊这个话题。我非常感谢娜娜的陪伴，这是最长情的告白。同时感谢吴怡编辑的理解与体谅，使得我还有机会完成此书。最后感谢开源社区，不仅帮助我成长，也提供了丰富的资料助力我完成此书，希望能有更多的机会回馈社区。

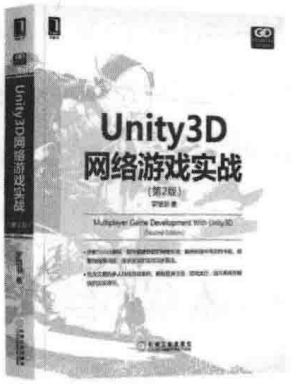
作者

于 2019 年元宵节

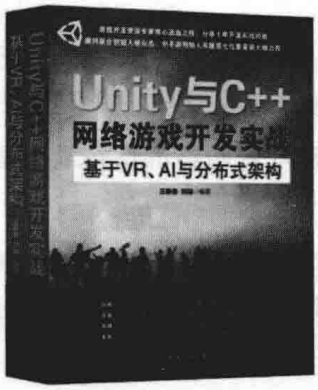
推荐阅读


游戏AI开发实用指南
Practical Game AI Programming


Unreal Engine 4
游戏开发秘笈
UE4虚拟现实开发
Unreal Engine VR Cookbook
Developing Virtual Reality with UE4

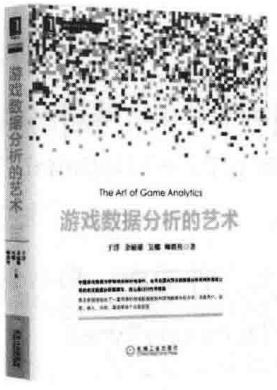

Unity3D
网络游戏实战
（第2版）
Multiplayer Game Development With Unity3D
Second Edition

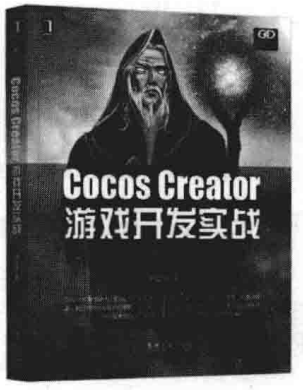

游戏服务器架构与优化
Architecture and Optimization for Game Servers


Unity与C++
网络游戏开发实战
基于VR、AI与分布式架构

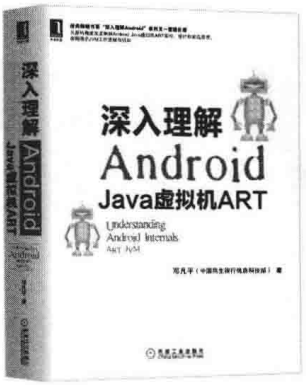

OpenGL编程指南
OpenGL Programming Guide


R语言游戏数据分析与挖掘


The Art of Game Analytics
游戏数据分析的艺术


Cocos Creator
游戏开发实战

推荐阅读


深入理解Android
Java虚拟机ART
Understanding Android Internals
ART JVM

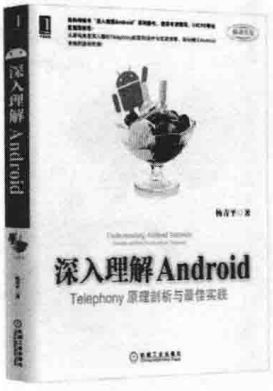

深入理解Android网络编程
技术详解与最佳实践


深入理解Android
Wi-Fi、NFC和GPS卷


深入理解Android
卷III

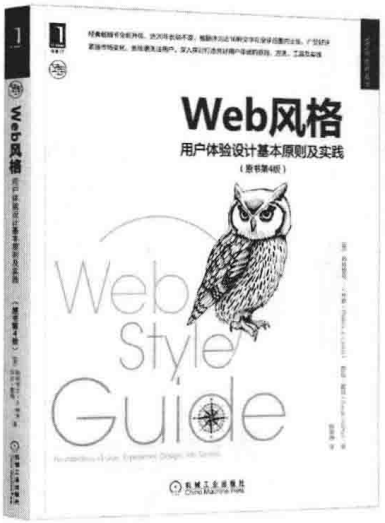

深入理解Android自动化测试
深入理解 Android
自动化测试


深入理解Android
WebKit 卷


深入理解Android
Telephony 原理剖析与最佳实践

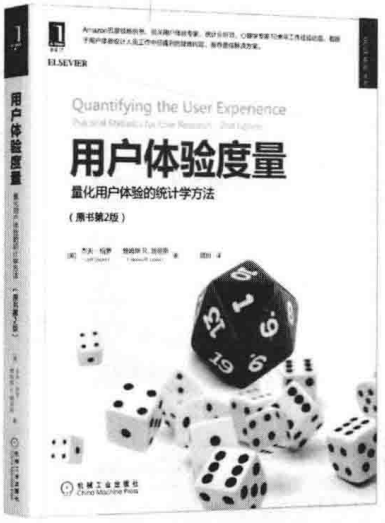
推荐阅读

UI/UE系列丛书



web风格:用户体验设计基本原则及实践（原书第4版）

ISBN：978-7-111-60798-4 定价：129.00元

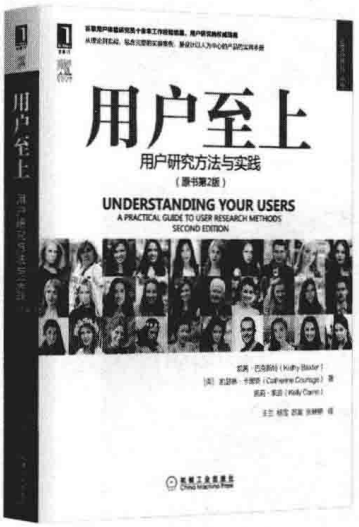


用户体验度量：量化用户体验的统计学方法（原书第2版）

ISBN：978-7-111-58965-5 定价：79.00元

用户体验乐趣多

ISBN：978-7-111-58693-7 定价：59.00元



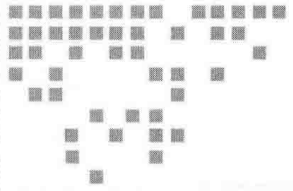
用户至上：用户研究方法与实践（原书第2版）

ISBN：978-7-111-56438-6 定价：99.00元

Contents 目　录

实战篇

基 础 篇



Chapter 1 第 1 章

Hello TypeScript

对于大部分前端开发者而言，TypeScript 是一门新鲜且陌生的语言。既然已经有 JavaScript 了，为什么还需要 TypeScript 呢？TypeScript 与 JavaScript 又有什么不同呢？

从某种严格的角度来讲，人们并不需要重复的东西，人们需要不一样的、特别的东西，就像两位面试者站在面试官前，都努力地表现出自己的差异性。

本章通过回溯 JavaScript 的发展历史来讲述 TypeScript 的差异性，希望通过梳理 TypeScript 的发展脉络，让读者了解只属于 TypeScript 的特性。

准备好了吗？就从 Hello TypeScript 开始吧。

1.1 引言

人们在使用一门语言之前需要对其进行多方“面试”。

你很难想象现在的前端世界竞争有多么激烈，除了长年稳坐第一的 JavaScript 以外，

还有 CoffeeScript、Dart、ClojureScript，等等，甚至“新人”Kotlin 也希望来蹭一下热度，分一杯羹。

因此，非常有必要先来看一下 TypeScript 的“个人简历”：

- JavaScript 的超集。
- 支持 ECMAScript 6 标准，并支持输出 ECMAScript 3 / 5 / 6 标准的纯 JavaScript 代码。
- 支持 ECMAScript 未来提案中的特性，比如异步功能和装饰器。
- 支持类型系统且拥有类型推断。
- 支持运行在任何浏览器、Node.js 环境中。

从这份简历可以了解到，TypeScript 与 JavaScript、ECMAScript 有着非常深入的联系。那么在谈论 TypeScript 之前，有必要对 JavaScript 与 ECMAScript 做一次充分的背景调查。也只有在了解了 JavaScript 与 ECMAScript 之后，我们才更有资格去谈论 TypeScript 是否适合我们。

1.1.1 JavaScript 与 ECMAScript

JavaScript 是蹭热点的“网红”出身。在 1995 年，它被搭载在网景浏览器中首次发布，当时名字还是 LiveScript。由于网景觉得这个名字缺乏热度，所以决定蹭一下流行的 Java 的热度，最终改名叫 JavaScript。

如果非要说 JavaScript 与 Java 有什么关系的话，大概就相当于雷锋与雷峰塔的关系了。在国外程序员圈内，则喜欢用 HAM（火腿）和 HAMSTER（仓鼠）来比喻两者的关系，如图 1-1 所示。

JavaScript 的成功引起了微软的注意，其在 IE 3.0 上搭载了 JScript。JScript 也是一种 JavaScript 的实现，两种 JavaScript 语言的出现意味着浏览器端语言标准化的缺失，甚至可能进一步出现分裂状态。

图 1-1　Java 与 JavaScript 的关系就像 HAM 之与 HAMSTER 之间的关系

所以在 1996 年，网景将 JavaScript 提交给 ECMA International（欧洲计算机制造商协会）进行标准化，最终确定了新的语言标准，取名为 ECMAScript。

从此，所有 JavaScript 的实现都必须以 ECMAScript 标准为基础。但由于 JavaScript 的历史原因，我们仍然用 JavaScript 称呼这门语言，而用 ECMAScript 称呼标准。

事情在开端的时候总是美妙的。

1997 年 ECMAScript 发布了首版标准，紧接着 1998 年 6 月发布第二版标准。但在 1999 年 12 月发布了第三版标准之后，不幸开始降临了。在接下来的 10 年里，ECMAScript 再也没能为标准化做出太大的贡献，甚至不同浏览器中的实现与标准大相径庭。不仅如此，各大厂商也开始向自己的 JavaScript 里添加“私货”，比如 JScript 的 ActiveXObject。

这 10 年里究竟发生了什么呢？比较公允的看法是由于 ECMAScript 4 过分激进的草案导致了浏览器厂商的一致抵制。IE 和 Flash 在这一时期的强势也导致了 ECMAScript 的进一步没落。

直到 2005 年秋，Task Group 1 of Ecma Technical Committee 39 (TG1) 才开始定期召

开会议。之后，大火的 Ajax 让人们意识到了 JavaScript 的复兴，标准化工作才开始加速。

经过一系列复杂的争论，2009 年 12 月 ECMAScript 5 得以发布。随后的 2012 年，国外的开发者社区推动停止对旧版本 IE 的支持工作，使得 ECMAScript 5 开始流行。

2015 年，ECMAScript 规范草案的委员会 TC39 决定将定义新标准的制度改为一年一次。这意味着 ECMAScript 的更新不再依赖于整个草案的完成度，而可以根据添加的特性进行滚动发布。

同年，代号为 Harmony 的 ECMAScript 6，也就是耳熟能详的 ES 6，或者叫 ES 2015，得以发布。Harmony（和谐）这个名字很有意思，仿佛在告诉开发者这么多年的争执与混乱终于平息。

但现在浏览器又开始拖后腿了。新特性往往很难在第一时间得到浏览器的支持，所以这一时期诞生了大量的前端工具，使开发者可以在开发环境中提前使用 ECMAScript 已发布或者还是草案的新特性。比如，Babel 通过插件化的方式引入 ECMAScript 的特性，并在生产环境时编译到 ES 3 或 ES 5 的代码。

1.1.2 TypeScript

TypeScript 是在新时代诞生的。

Ajax 的火热和 JavaScript 的复兴标志着一个全新时代的到来。这个时期的 JavaScript 代码正在变得越来越庞大，构建规模化 JavaScript 应用程序的需求日益旺盛。

微软的语言开发者发现，内部的研发部门以及外部的客户都表示 JavaScript 大型 Web 应用很容易出现失控，变得难以驾驭。而类似 CoffeeScript 和 Script# 的语言又难以使用 JavaScript 的语言特性。

微软认为 JavaScript 只是一门脚本语言，设计理念简单，缺乏对类与模块的支持，并非真正用于开发大型 Web 应用。这使得微软内部开始出现需要自定义工具去强化